

3주차 논문스터디 정리

An introduction to ROC analysis

Fawcett, Tom. "An introduction to ROC analysis." *Pattern Recognition Letters*, vol. 27, no. 8, 2006, pp. 861-874.

Reviewers: 고경현, 주혜인, 홍현경

1. Introduction

ROC(Receiver Operating Characteristics) 그래프는 성능에 따라 분류기를 시각화, 구성 및 선택하기 위한 도구이다.

ROC 그래프는 신호 감지 분야 또는 진단 검사 등에서 정상 감지 및 오감지의 관계를 파악하기 위해 쓰이기도 하고, 머신러닝 분야에서 알고리즘들의 성능을 평가하고 비교하기 위해서도 사용된다.

특히 오늘날에는 분류 문제에서의 단순 accuracy가 좋지 않은 평가 지표라는 인식으로 인하여 ROC 그래프가 머신러닝 분야에서 다수 사용되고 있다. 더불어 ROC 그래프는 분류 문제에서 클래스의 분포가 균일하지 못하고 치우친 경우, 그리고 오분류의 비용이 균등하지 않은 경우에 유용하다는 특징을 가지고 있어 중요성을 가진다.

2. Classifier performance(*in binary classification problems*)

이 섹션에서는 이진 분류 문제를 다룬다. 각 인스턴스 I 는 클래스 라벨을 나타내는 집합 $\{p, n\}$ 의 원소 중 하나에 매핑되어 있다. 여기에서 p 는 positive 클래스, n 은 negative 클래스를 나타내며, 이들은 예측값이 아닌 실제 인스턴스가 갖는 라벨이다.

분류 모델(또는 분류기)은 인스턴스들을 예측 클래스로 매핑하는 역할을 한다. 즉, 분류 모델은 인스턴스가 입력값으로 들어오면 그 인스턴스의 라벨 값을 예측한다. 이 분류 모델은 연속형 수치를 결과로 반환할 수도 있고, 또는 이산형 클래스 라벨 예측값을 결과로 반환할 수도 있다. 분류 모델에서 연속형 수치가 결과로 반환된다면 클래스 판단의 기준이 되는 수치인 임계치(threshold)가 필요하다. 논문에서는 인스턴스가 갖는 실제 클래스 라벨과 예측된 클래스 라벨을 구분하기 위해 실제 클래스 라벨은 $\{p, n\}$ 로, 예측된 클래스 라벨은 $\{Y, N\}$ 으로 나타낸다.

		실제값	
		p	n
예측값	Y	True Positives (TP)	False Positives (FP) : type I error
	N	False Negatives (FN) : type II error	True Negatives (TN)
열 합계		P	N

Confusion Matrix (= 2X2 Contingency Table, 혼동 행렬)

이진 분류 문제이므로 가능한 실제 클래스 값과 예측값은 각각 2개씩 존재한다. 따라서, 가능한 {실제값, 예측값} 조합은 $\{p, Y\}, \{p, N\}, \{n, Y\}, \{n, N\}$ 총 4가지이다. 이를 통해 위와 같은 혼동 행렬을 구성할 수 있으며, 이 혼동 행렬은 여러 지표의 기반이 된다. 지금부터는 혼동 행렬로부터 도출 가능한 지표들을 소개하고자 한다.

2.1 True Positive Rate(= TPR, recall, sensitivity)

우선 True Positive Rate(TPR)이다. TPR은 다른 말로 hit rate 또는 recall, sensitivity로 나타낸다. TPR은 실제 positive 클래스를 갖는 인스턴스들 중에서 positive로 예측이 된 인스턴스들의 비율이다. TPR을 구하는 식은 아래와 같이 나타낼 수 있다.

$$TPR (= recall, sensitivity) \approx \frac{TP}{TP + FN} = \frac{TP}{P}$$

즉, 위의 혼동행렬에서 첫 번째 열의 합을 분모에, 그 중 TP를 분자에 두면 TPR을 구할 수 있다.

2.2 False Positive Rate(= FPR)

다음은 False Positive Rate(FPR)이다. FPR은 다른 말로 false alarm rate라고도 한다. FPR은 실제 negative 클래스를 갖는 인스턴스들 중에서 negative로 예측이 된 인스턴스들의 비율이다. FPR을 구하는 식은 아래와 같이 나타낼 수 있다.

$$FPR \approx \frac{FP}{TN + FP} = \frac{FP}{N}$$

즉, 위의 혼동행렬에서 두 번째 열의 합을 분모에, 그 중 FP를 분자에 두면 FPR을 구할 수 있다.

2.3 Specificity

민감도인 sensitivity는 전체 P 중 TP에 관심을 가지고, $TPR = recall = sensitivity$ 의 관계를 가진다. 이는 앞에서 이미 언급을 했기 때문에 따로 소제목을 분류해서 설명을 하지는 않겠다.

특이도인 specificity는 전체 N중 TN에 관심을 가진다. Specificity를 구하는 식은 아래와 같다.

$$specificity = \frac{TN}{FP + TN} = 1 - FPR$$

2.4 Precision

Recall은 앞에서 계속 언급했듯이 TPR, sensitivity와 동일하므로 따로 설명은 하지 않겠다.

Precision은 recall과 동일하게 분자에 TP가 온다. 그러나, 분모에 실제 positive 클래스를 갖는 인스턴스의 수가 들어갔던 recall과는 달리, precision에서는 분모에 예측값이 positive 클래스인 인스턴스의 수가 들어간다. 즉, precision은 예측값이 positive인 인스턴스들 중에서 실제 값이 positive인 인스턴스들의 비율에 관심을 갖는다. Precision을 구하는 식은 아래와 같다.

$$precision = \frac{TP}{TP + FP}$$

2.5 Accuracy

Accuracy는 전체 인스턴스들 중에서 예측값과 실제값이 동일한 인스턴스들의 비율에 관심을 갖는다. Accuracy를 구하는 식은 아래와 같다.

$$accuracy = \frac{TP + TN}{P + N}$$

2.6 F-measure(= F1-score)

F-measure은 다른 말로 F1-score라고도 한다. F-measure은 precision과 recall의 조화평균이며, 이를 구하는 식은 아래와 같다.

$$F - measure = \frac{2}{1/precision + 1/recall}$$

3. ROC space

ROC 그래프는 FPR을 X축, TPR이 Y축으로 하는 2차원의 그래프로 true positives(= benefits)와 false positives(=costs)의 상대적인 tradeoff 관계를 나타낸다.

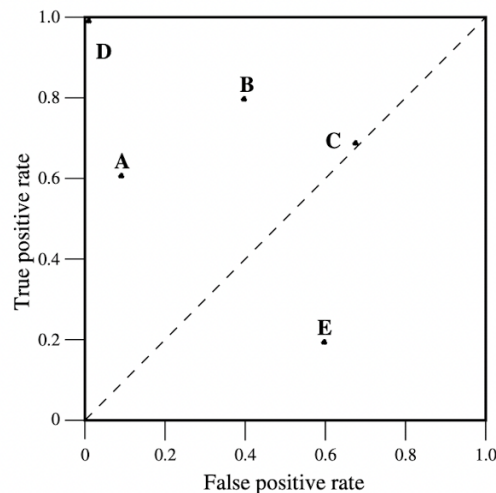


그림 출처: Fawcett Tom, 2006

다음은 이산형 클래스 라벨을 반환하는 분류 모델들을 나타낸 ROC 그래프이다. 각 분류 모델은 위의 그래프에 표현된 점에 해당하는 (FPR, TPR) 쌍을 하나씩 반환한다.

위의 그래프에서 $(0, 0)$ 은 $FPR = 0$, $TPR = 0$ 이므로 어떠한 경우에도 positive를 반환하지 않는 경우에 해당한다. 이와 반대로 $(1, 1)$ 은 $FPR = 1$, $TPR = 1$ 이므로 무조건적으로 positive만을 반환하는 경우에 해당한다. 더불어, $(0, 1)$ 에 해당하는 점 D는 $FPR = 0$ 이고 $TPR = 1$ 이므로 완전한 분류에 해당한다.

ROC 공간에서 두 점을 비교하는 상황을 가정해 볼 때, 두 점 중 북서쪽에 위치하는 분류 모델의 성능이 더 좋다고 할 수 있다. 북서쪽에 위치한다는 것은 FPR이 더 작고, TPR이 더 크기 때문이다.

ROC 공간 상에서 어떤 점이 공간 상에서 왼쪽, X축에 가깝게 위치한다면 그 분류 모델은 conservative하다고 한다. 이와 반대로 공간 상에서 우측 위에 위치하는 경우에는 해당 분류 모델이 liberal하다고 한다. 실생활에서는 negative인 인스턴스들이 지배적이라는 점을 생각해 보면 ROC 공간 상에서 아주 왼쪽에 위치하는 분류 모델의 경우에는 주목해 볼 만 하다.

3.1 Random performance

위의 그림에 나타난 ROC 공간 상에서 점선은 $y = x$ 를 나타내는데, 이는 클래스를 랜덤으로 추측하는 경우에 해당한다. 따라서 클래스를 랜덤으로 추측하는 분류 모델의 경우에는 positive로 분류하는 빈도가 어떠한지에 따라 $y = x$ 직선 상에서 움직이게 된다. 위의 그림에서 점 C는 $y = x$ 상에 위치하며, $(0.7, 0.7)$ 의 좌표를 갖는다. 따라서, 점 C에 해당하는 분류

기는 전체의 70%를 positive로 추측한다. 이러한 분류 모델들의 경우에는 결과를 단순히 추측하는 것이기 때문에 유용하지 않다.

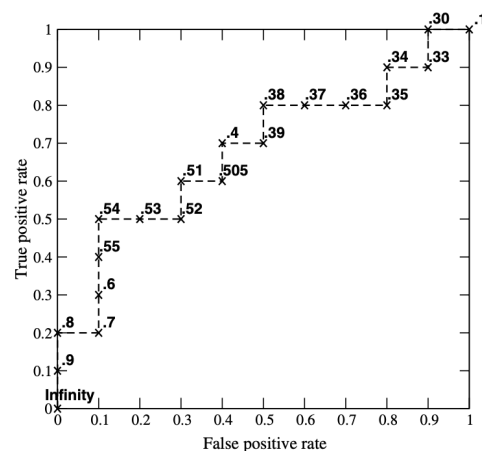
앞서 말한 바와 같이, $y = x$ 는 클래스를 랜덤으로 추측하는 경우에 해당하기 때문에 이보다 FPR이 높고 TPR이 낮은 분류 모델들, 즉 $y = x$ 직선 아래에 위치하는 분류 모델들의 경우에는 랜덤으로 추측을 하는 것보다 더 좋지 않은 성능을 가진다고 할 수 있다. 이 때, $y = x$ 직선 아래에 위치하는 분류 모델이 예측한 값을 반대로 가져간다면 $y = x$ 직선 위의 성능을 얻을 수 있다. 이러한 이유에서 $y = x$ 직선 아래에 위치하는 모델이라도 유용한 정보를 지닐 수 있다.

4. Curves in ROC space

앞서 언급한 바와 같이, 분류 모델은 이산형 클래스 라벨을 결과로 반환할 수도 있고, 연속형 수치를 결과로 반환할 수 있다. 만약 분류 모델이 이산형 클래스 라벨을 반환한다면 이 분류 모델은 ROC 공간 상에서 하나의 점만을 나타내게 된다. 어떤 인스턴스가 특정 클래스에 속할 정도에 대한 연속형 수치를 반환하는 분류 모델의 경우에는 임계값이 필요하다. 이 임계값보다 인스턴스에 대한 수치가 크다면 Y 를, 그렇지 않은 경우에는 N 을 반환하여 이산형 클래스를 최종적으로 반환한다. 이 때, 각 임계값이 달라질 때 마다 ROC 공간에 서로 다른 점들이 찍히게 된다.

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

그림 출처: Fawcett Tom, 2006



왼쪽의 표는 20개의 인스턴스들로 구성된 테스트셋을 이용하여 score를 구한 결과이고, 오른쪽 그래프는 임계값을 변경해가며 ROC 곡선을 그린 결과이다. 그래프에서 알 수 있는 바와 같이 임계값이 바뀔 때 마다 ROC 공간 상에 서로 다른 점이 하나씩 찍혀 나간다. 이 예시에서는 유한개의 인스턴스들을 이용했기 때문에 ROC 곡선이 계단 함수 형태이지만, 인스턴스의 수를 무한대로 늘리게 되면 일반적인 곡선 형태가 된다.

임계값이 작게 되면 더 많은 인스턴스들을 positive로 예측하게 된다. 따라서, FPR, TPR 모두 늘어가게 되어 ROC 공간 상의 우측 위에 점이 찍히게 된다. 달리 말하면, 임계값을 작게 할수록 conservative한 분류 모델이 liberal해진다.

4.1 Relative versus absolute scores

ROC 그래프는 negative와 positive 인스턴스를 구별하기 위해 상대적인 인스턴스의 점수를 구한다는 특징을 가진다.

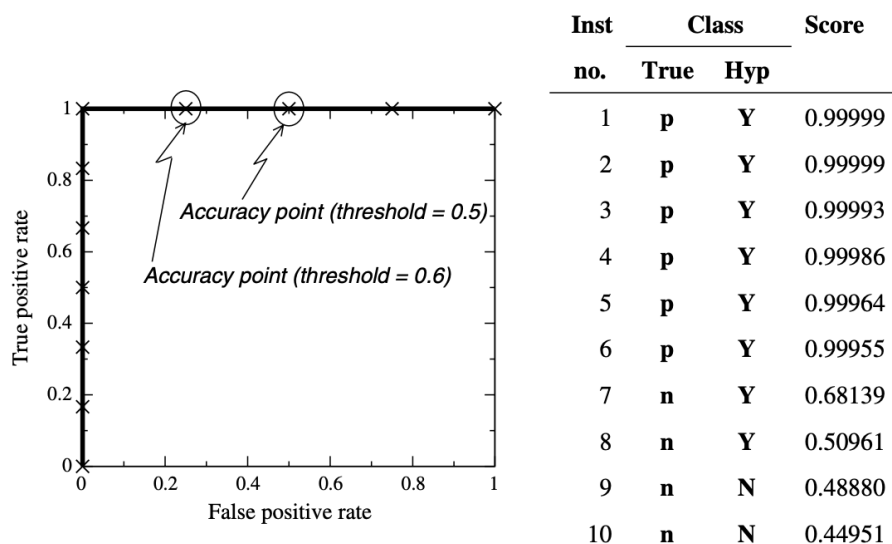


그림 출처: Fawcett Tom, 2006

다음의 그림과 표를 보자. 임계값을 0.5로 잡았을 때에는 7번과 8번 인스턴스를 오분류하기 때문에, accuracy는 80%가 된다. 그러나, accuracy가 80%임에도 불구하고 ROC 곡선은 완전한 분류의 성능을 나타낸다.

이러한 accuracy와 ROC 곡선 상의 차이는 각각이 무엇을 측정하고 있는지가 다르기 때문에 발생한다. ROC 곡선은 FPR 대비 TPR을 구한다. 즉, 어떤 분류 모델이 negative인 인스턴스들에 비해 positive인 인스턴스들을 rank하는 성능을 나타내는 것이다. 그러나, accuracy는 임계값을 이용하여 절대적인 점수에 따른 분류 결과를 측정한다.

4.2 Class skew

ROC 곡선은 클래스 분포의 변화에 영향을 받지 않아서, positive인 클래스와 negative인 클래스의 비율이 달라지더라도 ROC 곡선 자체는 변하지 않는다. 이 이유는 앞에서 언급한 confusion matrix를 보면 알 수 있다.

		실제값	
		p	n
예측값	Y	True Positives (TP)	False Positives (FP) : type I error
	N	False Negatives (FN) : type II error	True Negatives (TN)
열 합계		P	N

Confusion Matrix (= 2X2 Contingency Table, 혼동 행렬)

ROC 공간의 X좌표는 FPR, Y좌표는 TPR이다. 즉, ROC 곡선은 FPR과 TPR에 기반한다. 이 때, TPR을 계산할 때에는 혼동행렬의 왼쪽 칼럼만 이용하게 되고, FPR을 계산할 때에는 혼동행렬의 오른쪽 칼럼만을 이용하게 된다. 즉, FPR과 TPR을 구할 때에는 각각 서로 다른 칼럼 하나씩만을 이용하기 때문에 테스트셋에서 positive : negative가 변하더라도 ROC 곡선은 변하지 않는다. 이러한 이유에서 ROC 곡선은 클래스의 분포 변화에 민감하지 않다고 할 수 있다.

이와 달리 accuracy, precision, F-score 등을 계산하기 위해서는 양 쪽 칼럼을 모두 이용해야 한다. 이와 같이 양쪽 칼럼을 모두 이용하는 평가 지표들은 클래스의 분포에 민감할 수 밖에 없고, 따라서 클래스의 분포가 달라지면 계산 결과 역시 달라진다.

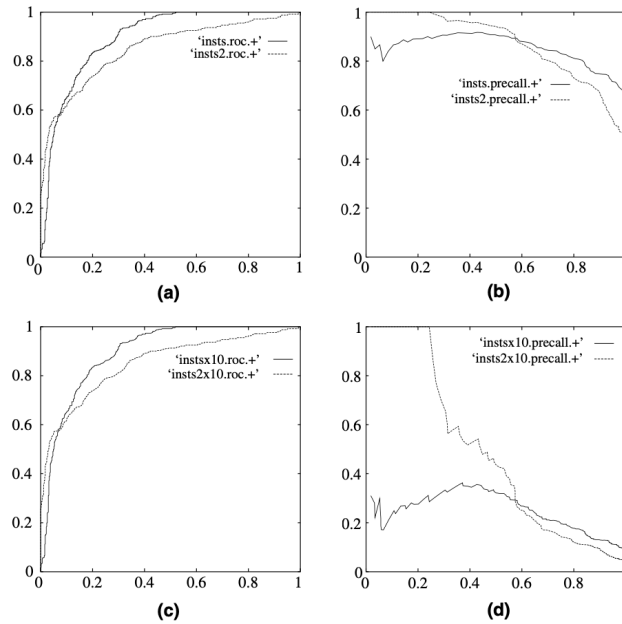


Fig. 5. ROC and precision-recall curves under class skew. (a) ROC curves, 1:1; (b) precision-recall curves, 1:1; (c) ROC curves, 1:10 and (d) precision-recall curves, 1:10.

사진 출처: Fawcett Tom, 2006

혼동행렬에서 양 쪽 칼럼을 모두 사용하는 경우는 클래스 분포 변화에 민감하고, 한 쪽 칼럼을 이용하는 경우는 클래스 분포 변화에 민감하지 않다는 사실은 위의 그래프를 통해서도 확인할 수 있다.

위의 그래프에서 (a)와 (c)는 ROC 곡선을, (b)와 (d)는 precision-recall 곡선을 나타낸다. 더불어 (a)와 (b)에서는 positive : negative = 1 : 1이고, (c)와 (d)에서는 positive : negative = 1 : 10이다. (a)와 (c)를 보면 클래스의 분포가 달라졌음에도 불구하고 ROC 곡선의 변화가 없음을 알 수 있다. 이와 달리 (b)와 (d)는 혼동행렬에서 양 쪽 칼럼을 모두 사용하기 때문에 클래스 분포가 달라지자 곡선의 형태 역시 달라진 것을 확인할 수 있다.

4.3 Creating scoring classifiers

많은 분류 모델들은 각 인스턴스들에 대하여 이산형 클래스 라벨만을 반환한다. 그러나, 이러한 이산형 분류 모델들도 그 모델들이 가지고 있는 인스턴스들에 대한 통계량을 이용하면 점수(score, 연속형 수치)를 만들어낼 수 있다. 의사 결정 나무의 리프 노드에 대한 클래스 비율 등이 그 예시이다.

또한, 클래스 라벨만을 반환하는 분류 모델들을 합침으로써 점수를 산출해 낼 수 있다. 논문에서는 모델 앙상블을 위한 배깅(bagging)을 예시로 들면서, voting 결과의 집합을 점수를 산출하는데 이용할 수 있음을 밝힌다. 더불어 scoring과 voting을 결합한 방식 역시 사용할 수 있음을 언급하고 있다.

이처럼 ROC curve는 클래스 불균형이 있어도 다른 지표들에 비해 robust하며, discrete한 예측을 하는 모델에 대해서도 적용 가능하다는 장점이 존재한다.

5. Efficient generation of ROC curves

그렇다면 ROC curve를 어떻게 효율적으로 그릴 수 있을까? 우리는 threshold에 의해 분류를 할 때 갖는 monotonicity 특징을 이용해서 ROC curve를 효율적으로 그릴 수 있다.

- monotonicity

예를 들어, 로지스틱 회귀 모델에서 예측된 y (positive class에 속할 확률)가 0.7 이상인 경우에 한하여 positive로 분류한다고 가정하자. 이때 가정했던 threshold에 해당하는 0.7보다 threshold가 낮아지더라도 기존에 positive class로 분류되었던 관측치는 여전히 positive class로 분류된다는 특징이 바로 monotonicity다.

그래서 우리는 ROC curve를 그릴 때 score(= predicted probability)를 내림차순으로 정렬한 후 threshold를 점차 낮춰가며 True Positive와 False Positive를 누적하고, 이를 이용해 ROC curve를 그린다. 이것을 설명한 것이 아래 알고리즘이다.

Algorithm 1

Algorithm 1. Efficient method for generating ROC points
Inputs: L , the set of test examples; $f(i)$, the probabilistic classifier's estimate that example i is positive; P and N , the number of positive and negative examples.
Outputs: R , a list of ROC points increasing by fp rate.
Require: $P > 0$ and $N > 0$

```

1:  $L_{sorted} \leftarrow L$  sorted decreasing by  $f$  scores
2:  $FP \leftarrow TP \leftarrow 0$ 
3:  $R \leftarrow \langle \rangle$ 
4:  $f_{prev} \leftarrow -\infty$ 
5:  $i \leftarrow 1$ 
6: while  $i \leq |L_{sorted}|$  do
7:   if  $f(i) \neq f_{prev}$  then
8:     push  $\left(\frac{FP}{N}, \frac{TP}{P}\right)$  onto  $R$ 
9:      $f_{prev} \leftarrow f(i)$ 
10:  end if
11:  if  $L_{sorted}[i]$  is a positive example then
12:     $TP \leftarrow TP + 1$ 
13:  else /*  $i$  is a negative example */
14:     $FP \leftarrow FP + 1$ 
15:  end if
16:   $i \leftarrow i + 1$ 
17: end while
18: push  $\left(\frac{FP}{N}, \frac{TP}{P}\right)$  onto  $R$  /* This is (1,1) */
19: end

```

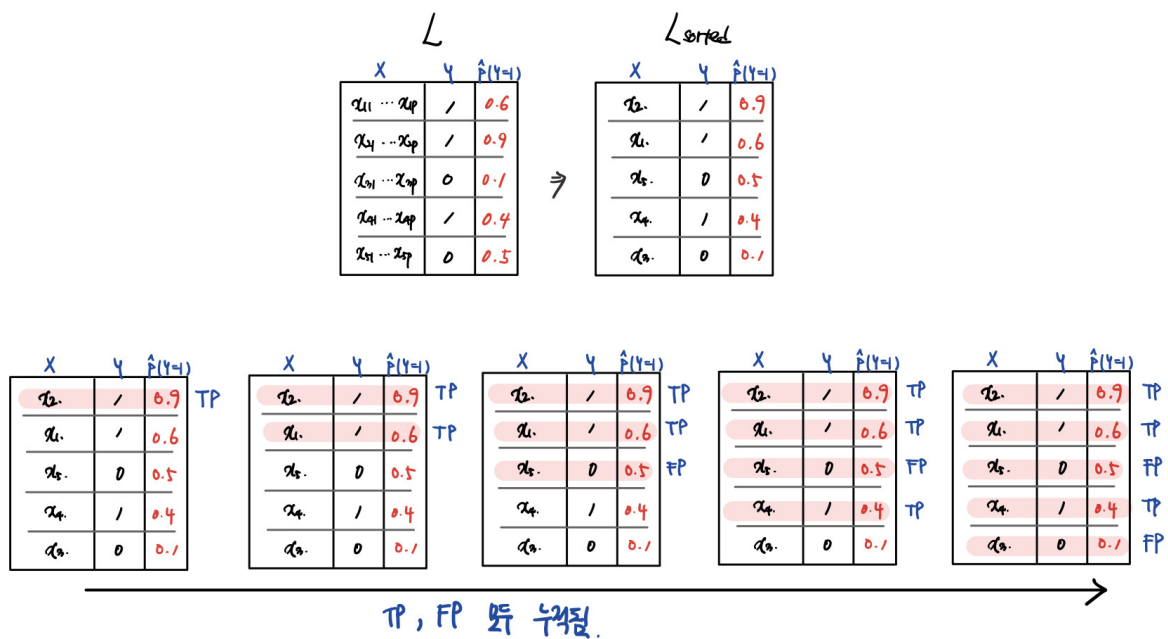
- Line 1 : 예측된 score에 따라 내림차순으로 test data를 정렬

- Line 2 - Line 5 : 기본값 할당
- Line 6 : 전체 반복문 시작
- **Line 7 - Line 10** : 해당 예측 score가 이전과 동일한 score가 아니라면 ROC space에 누적된 TP와 FP에 따라 좌표 할당
- **Line 11 - Line 16** : threshold에 따라 Positive로 분류된 obs가 TP인지 FP인지 분류하여 누적
- Line 18 : 좌표 (1,1)을 ROC space에 마지막으로 추가

실질적으로 ROC curve를 그리는 부분은 **Line 7 - Line 16** 이다.

• Line 11 - Line 16

먼저 Line 11 - Line 16을 아래 그림 예시로 설명할 수 있다.

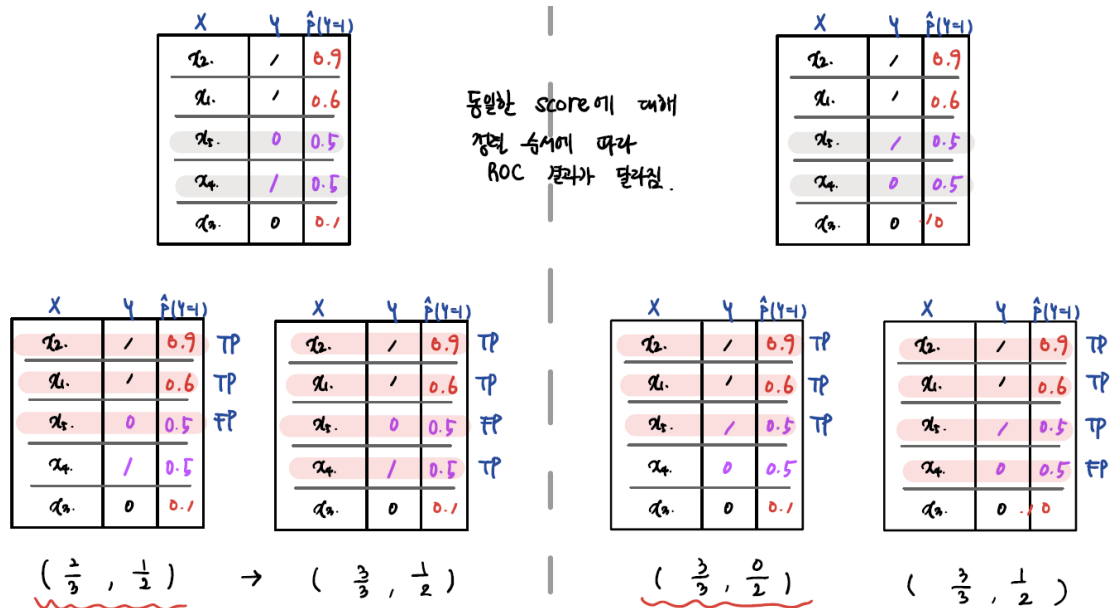


예측된 score에 의해 정렬된 L_{sorted} 로부터 각각의 score를 threshold로 삼아 TP와 FP를 구분해낸다. 빨간색 하이라이트가 된 것들은 threshold에 의해 positive (1)로 예측된 관측치이며 while이 돌아감에 따라 하나씩 추가된다. 빨간색 하이라이트가 없는 것들은 negative (0)로 예측된 관측치들인데 ROC curve를 그릴 때는 이 관측치들은 필요 없다. 왜냐하면 TPR과 FPR을 구할 때 negative로 예측된 관측치의 개수는 사용하지 않기 때문이

다. 따라서 위와 같이 단순한 방식으로 TP와 FP를 누적하고, **Line 8**에서 원래 데이터의 Positive개수 P 와 Negative 개수 N 으로 각각 나누어 TPR과 FPR을 계산한다.

• Line 7 - Line 10

다만 여기서 예측된 score가 같은 관측치가 존재한다면 ROC curve를 그릴 때 문제가 발생할 수 있다. 이 문제를 방지하는 부분이 Line 7 - Line 10인데 우선 어떤 문제가 발생하는지 아래 그림으로 보자.



동일한 score 0.5를 갖는 관측치가 두 개 있고, 각각 대응하는 실제 Y값은 0과 1로 서로 다르다. 알고리즘에 따라 이를 score에 따라 내림차순으로 정렬한다면 왼쪽과 오른쪽 위의 경우처럼 두 경우가 존재할 수 있다. 각각을 이용해 그림의 아래 부분처럼 ROC curve의 좌표를 계산한다면 서로 다른 결과를 도출하게 된다.

극단적으로 생각했을 때, 이같은 문제는 아래처럼 결과를 왜곡하는 ROC curve를 도출한다.

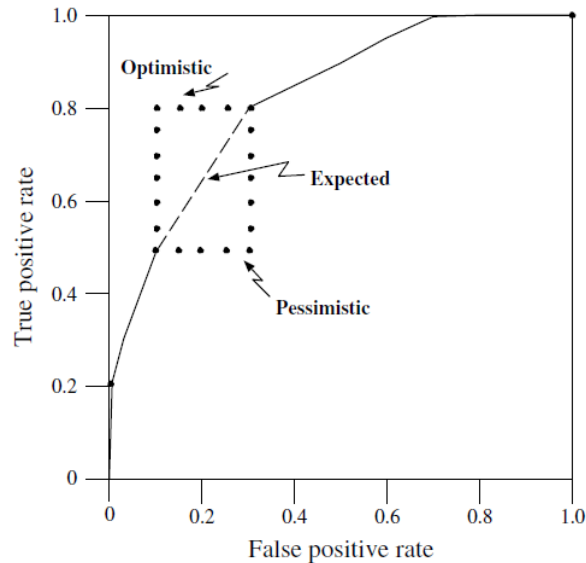


Fig. 6. The optimistic, pessimistic and expected ROC segments resulting from a sequence of 10 equally scored instances.

동일한 score에 대해 optimistic하게 ROC curve를 그려 실제보다 성능이 좋다고 과대평가할 수도 있으며, 반대로 pessimistic하게 그려 성능을 과소평가할 수 있다. 하지만 ROC curve는 분류 모델의 기대 성능 (expected performance)를 나타내야 하므로 두 경우의 평균치인 대각선으로 ROC curve를 그려야 한다.

따라서 논문에서는 Line 7 - Line 10 을 집어넣음으로써 위 문제를 해결하고자 했다. 해당 Line을 통해 동일한 score를 갖는 관측치에 해당하는 좌표는 처음과 끝만 나타내고 관측치 정렬 순서에 의해 바뀔 수 있는 중간 부분은 curve에 나타내지 않게 된다.

이렇게 예측된 score가 동일하게 나오는 경우는 종종 발생하는데, 그 예로 트리 모델에서 리프 노드에 있는 클래스의 비율이 있다. 동일한 리프 노드에 속한 관측치들은 모두 동일한 score를 갖게 되기 때문이다.

▼ 근데.. 위 알고리즘에서 드는 생각들

1. 같은 score에 해당하는 값이 정렬 순서에 의해 왜곡되는 것을 방지하기 위해 Line7~Line10을 넣었는데, 그럼 f 도 L 처럼 score에 따라 내림차순 해야하지 않나? 그렇게 하지 않으면, 같은 score를 갖더라도 서로 연속되어 있지 않다면 왜곡해서 ROC를 그리게 되는데..
2. 만약에 동일한 score를 갖더라도 그 실제 값들이 0또는 1로 모두 같다면, 오히려 논문에 등장한 방식이 결과를 왜곡하는 것이 되어버린다.

3. 애초에 관측치별(i 를 하나씩 증가시키면서)로 TP, FP를 누적하지 말고 unique한 score를 기준으로 TP와 FP를 구분한다면 Line7-Line10도 필요 없을 것 같다.

6. The ROC convex hull

ROC 그래프의 강점은 class distribution이나 error cost같은 operating condition에 상관 없이 일정하다는 것이다.

반대로 생각한다면, 우리가 특정 class를 예측하지 못하는 것에 더 큰 가중치를 주거나(error cost), 클래스 불균형이 존재하더라도 ROC graph 자체는 그것을 반영하지 않기 때문에 과연 어느 point의 분류 모델이 가장 좋은 성능을 나타내는지 알기 어렵다.

이를 보완하는 것이 바로 **iso-performance line**이다. *iso*-는 '동일한'이란 의미다.

- **iso-performance line**

이 line은 ROC space 상에서 특정 조건(class distribution / error costs) 하에서 가장 낮은 기대 비용(=가장 좋은 성능)을 나타내는 분류 모델을 찾는다.

iso-performance line의 기울기 m 은 아래와 같다.

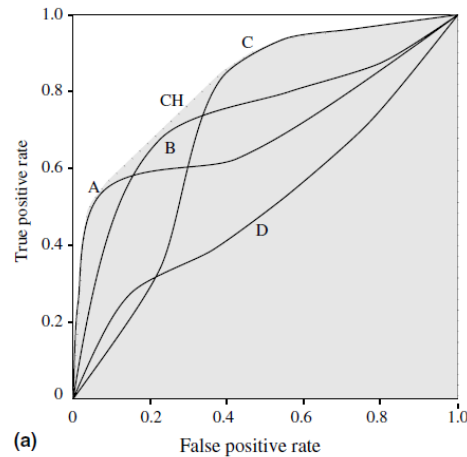
$$\frac{TP_2 - TP_1}{FP_2 - FP_1} = \frac{c(Y, n)p(n)}{c(N, p)p(p)} = m$$

- $c(Y, n)$: false positive cost
- $c(N, p)$: false negative cost
- $p(n)$: negative class의 비율
- $p(p)$: positive class의 비율

기울기가 cost의 비율과 class의 비율의 곱으로 표현되기 때문에 기울기가 같은 직선 위의 모든 분류 모델은 동일한 operating condition을 갖는다. 이 특징이 분류 모델을 찾는 데에 주요하게 작용한다.

iso-performance line을 이용한다면 특정 조건 하에서 best classifier를 찾을 수 있다고 했다. 이 방법은 아래 ROCCH를 설명한 후에 설명하도록 한다.

- ROC convex hull (ROCCH)

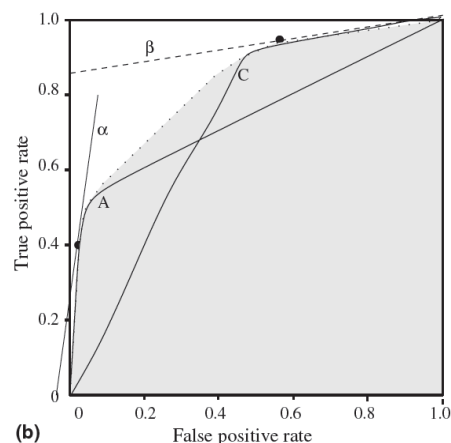


Convex hull은 특정 set을 모두 포함하게 만드는 최소한의 convex 집합을 의미한다. 이 Convex hull을 ROC space에 적용시킨 것이 바로 ROC convex hull이다.

위 그림을 보면 그 정의를 직관적으로 이해할 수 있다. 네 개의 classifier A, B, C, D가 있다. 이 classifier를 모두 포함시키는 최소한의 convex hull이 바로 CH라고 표기된 회색 영역이 된다.

그림을 보면 알 수 있듯이 ROCCH는 A와 C에 의해서만 bounded된다. 따라서 A와 C는 optimal하며, B와 D는 optimal하지 않다. 그렇기에 만약 우리가 최적의 분류 모델을 찾고자 할 때 B와 D는 고려대상에서 바로 제외된다.

이제 ROCCH와 iso-performance line을 이용해 특정 조건 하에 가장 최고의 성능을 갖는 classifier을 예시를 통해 찾아보자.



- Case 1 (α)

negative의 개수와 positive 개수가 10:1로 불균형이고, error cost는 1:1로 동일하다고 하고, 이 상황에서 위 ROCCH 상에서 가장 좋은 성능을 나타내는 분류 모델을 찾고자 한다.

특정 조건이 고정되므로 위 iso-performance line의 기울기를 구하는 식에 의해 기울기는 10으로 고정된다. 이 조건 하에 가장 좋은 성능을 나타내기 위해서는 TPR가 가장 크면 된다. 다시 말해서, ROCCH에서 10으로 고정된 기울기의 iso-performance line의 y절편이 가장 큰 접점을 찾으면 된다. 이는 위 그림에서 직선 α 가 된다.

- Case 2 (β)

다음으로 두 클래스의 비율은 동일하나 false negative가 false positive에 비해 10배 많은 cost를 부여한다고 가정하자.

이 조건 하에 iso-performance line의 기울기는 1/10으로 고정되고, 위 방식과 동일하게 이 조건 하에서 가장 좋은 성능을 나타내는 분류 모델을 찾고자 한다면 ROCCH 상에서 TPR이 가장 크게 나타나는 접점을 찾으면 된다. 이는 위 그림에서 직선 β 가 된다.

이처럼 iso-performance line은 ROC space에서 특정 조건 하에 가장 좋은 성능을 보이는 분류 모델을 찾는다. potentially optimal한 classifier만이 ROC convex hull 위에 존재하기 때문에, ROCCH 역시 optimal한 분류 모델을 찾는 데에 유용하다.

- **Interpolating classifiers** (논문에는 section 10으로 소개되어 있지만 흐름 상 변경)

만약 ROCCH 위에 있어 optimal하면서도 A와 C 사이에 위치한 분류 모델을 찾고 싶다면, 두 모델을 interpolation하는 방법도 존재한다.

구체적인 예시를 통해서 이해해보자.

새로운 보험을 제안할 마케팅 대상 4000명이 있다고 가정하자. 하지만 제한된 예산에 따라 정확히 800명에게만 광고가 가능하다고 한다면, 우리는 우리의 제안에 더 잘 응답할 것 같은 사람 800명을 선택하고자 할 것이다. 그렇다면, 우리의 목표는 **4000명 중 가장 잘 응답할 것 같은 800명을 먼저 분류**해내 그 사람들에게 마케팅 제안을 하는 것이다.

평균적인 응답률은 6%라고 한다면, 4000명 중 240명은 응답하고, 3760명은 그렇지 않을 것이라 생각할 수 있다.

두 분류 모델 A, B가 이미 있다고 했을 때, 모델 A는 ROC space 위에서 (0.1, 0.2)에 위치하고 B는 (0.25, 0.6)에 위치한다.

목적은 정확히 800명을 잘 응답할 것이라고 예상되는 positive class로 분류하여 마케팅을 하는 것이다. 따라서 제약 조건은

$$\text{FPR} \times 3760 + \text{TPR} \times 240 = 800$$

이 된다.

이 상황에서 만약 A로 분류한다면 positive로 분류되는 고객은 목표치보다 너무 적은 $0.1 \times 3760 + 0.2 \times 240 = 424$ 명이 되고, B로 분류한다면 목표치보다 너무 많은 $0.25 \times 3760 + 0.6 \times 240 = 1084$ 명이 된다.

따라서 우리는 A와 B 사이에 있는 모델을 찾아내어 정확히 800명의 고객을 분류해내고 싶다. 이때 바로 linear interpolation (선형 보간)이 사용된다.

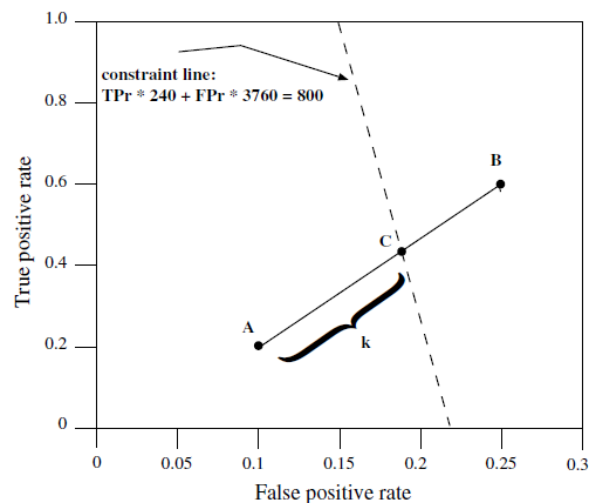


Fig. 10. Interpolating classifiers.

분류 모델 A,B와 제약 조건을 ROC space에 나타내면 위 그림의 점선과 같다. A,B 사이에 존재하면서 제약 조건을 만족시키는 모델은 C이고 C의 좌표는 대략 (0.18, 0.42)이다.

이때 interpolation은 위 좌표를 갖는 모델을 찾아내는 것이 아니라 기존에 알고 있는 A,B를 이용하여 C의 결과를 sampling 하는 것이다.

A와 B사이의 C에 대한 비율 k 는 다음과 같이 계산된다.

$$k = \frac{0.18 - 0.1}{0.25 - 0.1} \approx 0.53$$

따라서 B의 결과를 0.53의 비중으로 sampling하고 A의 결과를 0.47의 비율로 sampling해 그것을 예측 결과로서 활용한다면, 그것이 바로 우리가 찾고자 했던 모델 C의 결과가 되는 것이다.

실제로 이러한 fractional sampling은 다음과 같이 적용된다. 각각의 관측치에 0과 1사이의 random number를 할당한 후에 그 숫자가 k 보다 크면 관측치를 모델 A로 분류하고, 작다면 모델 B로 분류하는 식으로 interpolation이 이루어진다.

7. Area under an ROC curve (AUC)

여러개의 분류 모델의 성능을 비교하기 위해 ROC curve를 하나의 scalar 값으로 나타내기 위해 고안된 것이 AUC이다. 즉, AUC는 분류 성능을 비교하기 위한 일반적인 지표다.

- 기본 개념

AUC는 ROC space 위의 지표이기 때문에 항상 단위 사각형의 넓이 1과 0사이의 값이다. 또한 random한 분포로부터의 분류 모델 diagonal한 ROC curve를 그리므로 이때의 AUC는 0.5 (직각 삼각형의 넓이)가 된다. 즉, 현실적으로 random한 데이터를 분류하는 것보다 성능이 좋아야하므로 AUC는 0.5보다 작아서는 안된다.

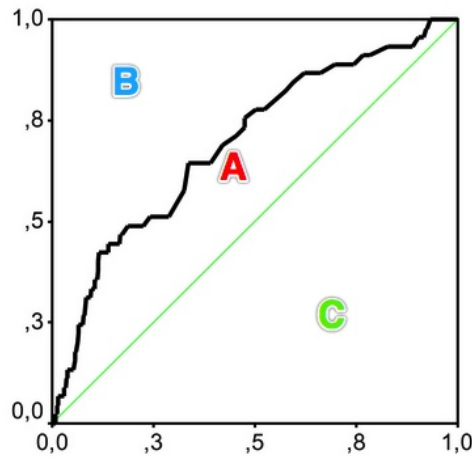
- 다른 개념과의 연관성

1. Wilcoxon test of ranks

AUC는 임의로 선택된 positive 관측치를 임의로 선택된 negative 관측치보다 높게 순위를 매길 확률과 동일하다. 이러한 관점에서 AUC는 Wilcoxon test of ranks의 test statistics과 동일하다고 할 수 있다.

2. Gini coefficient

지니 계수는 $y = x$ 과 x 축, y 축으로 둘러싸인 넓이 ($\frac{1}{2}$) 대비 AUC와 $y = x$ 사이의 넓이의 비율과 같다. 이 사실을 이용해 지니 계수와 AUC 간의 관계를 도출할 수 있다.

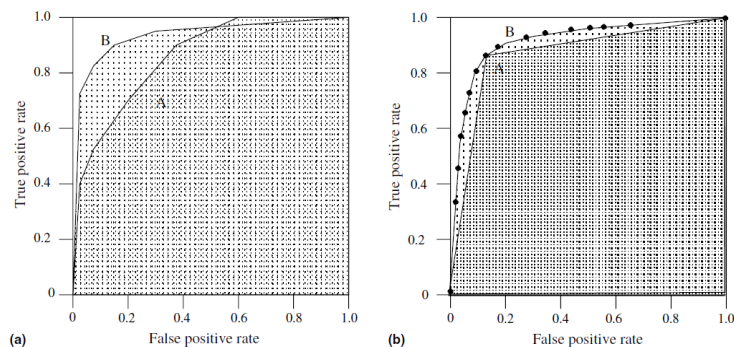


$$G = \frac{A}{A+B} = A/0.5 = 2A$$

$$AUC = A + C = A + 0.5$$

$$\therefore 2AUC = G + 1$$

- 특징



오른쪽 그림의 경우 모델 B가 A보다 AUC가 크므로 성능이 더 좋다고 할 수 있다.

AUC가 일반적인 평가지표로서 좋은 것은 사실이지만, 왼쪽 그림을 보면 더 큰 AUC가 무조건 좋은 성능을 보장하는 것은 아니라는 걸 알 수 있다. AUC는 B가 더 크지만, false positive rate가 0.6 이상인 부분에서는 A의 성능이 더 좋기 때문이다.

AUC를 계산하는 알고리즘은 다음과 같다.

Algorithm 2

Algorithm 2. Calculating the area under an ROC curve
Inputs: L , the set of test examples; $f(i)$, the probabilistic classifier's estimate that example i is positive; P and N , the number of positive and negative examples.
Outputs: A , the area under the ROC curve.
Require: $P > 0$ and $N > 0$

```

1:  $L_{\text{sorted}} \leftarrow L$  sorted decreasing by  $f$  scores
2:  $FP \leftarrow TP \leftarrow 0$ 
3:  $FP_{\text{prev}} \leftarrow TP_{\text{prev}} \leftarrow 0$ 
4:  $A \leftarrow 0$ 
5:  $f_{\text{prev}} \leftarrow -\infty$ 
6:  $i \leftarrow 1$ 
7: while  $i \leq |L_{\text{sorted}}|$  do
8:   if  $f(i) \neq f_{\text{prev}}$  then
9:      $A \leftarrow A + \text{TRAPEZOID\_AREA}(FP, FP_{\text{prev}},$ 
        $TP, TP_{\text{prev}})$ 
10:     $f_{\text{prev}} \leftarrow f(i)$ 
11:     $FP_{\text{prev}} \leftarrow FP$ 
12:     $TP_{\text{prev}} \leftarrow TP$ 
13:   end if
14:   if  $i$  is a positive example then
15:      $TP \leftarrow TP + 1$ 
16:   else /*  $i$  is a negative example */
17:      $FP \leftarrow FP + 1$ 
18:   end if
19:    $i \leftarrow i + 1$ 
20: end while
21:  $A \leftarrow A + \text{TRAPEZOID\_AREA}(N, FP_{\text{prev}}, N, TP_{\text{prev}})$ 
22:  $A \leftarrow A / (P \times N)$  /* scale from  $P \times N$  onto the unit square */
23: end

1: function  $\text{TRAPEZOID\_AREA}(X1, X2, Y1, Y2)$ 
2:    $\text{Base} \leftarrow |X1 - X2|$ 
3:    $\text{Height}_{\text{avg}} \leftarrow (Y1 + Y2) / 2$ 
4:   return  $\text{Base} \times \text{Height}_{\text{avg}}$ 
5: end function

```

ROC curve를 그리는 Algorithm 1과 꽤나 유사한데, 넓이를 구하는 Line 9, Line 21-22와 알고리즘 아래쪽에 ROC curve에 의해 그려지는 사다리꼴 넓이를 구하는 함수 `TRAPEZOID_AREA`가 추가되었다.

전반적인 과정은 ROC curve 알고리즘과 마찬가지로 예측 score에 따라 정렬된 데이터로 TP와 FP를 누적시켜나간다. 누적된 TP 와 FP , 그리고 이전 누적값인 TP_{prev} 를 이용해 FP_{prev} 그려지는 사다리꼴의 넓이를 구하고 이것을 거듭하여 더하게 된다.

제시된 알고리즘에서 주의해야할 점은 **Line 22**이다. ROC space 상에서 바로 AUC를 계산하는 것이 아닌, TP와 FP의 count 값으로 AUC를 계산한 후에 마지막에 기존 class의 개수로 scaling하여 0과 1 사이의 값으로 표현한다.

8. Averaging ROC curves

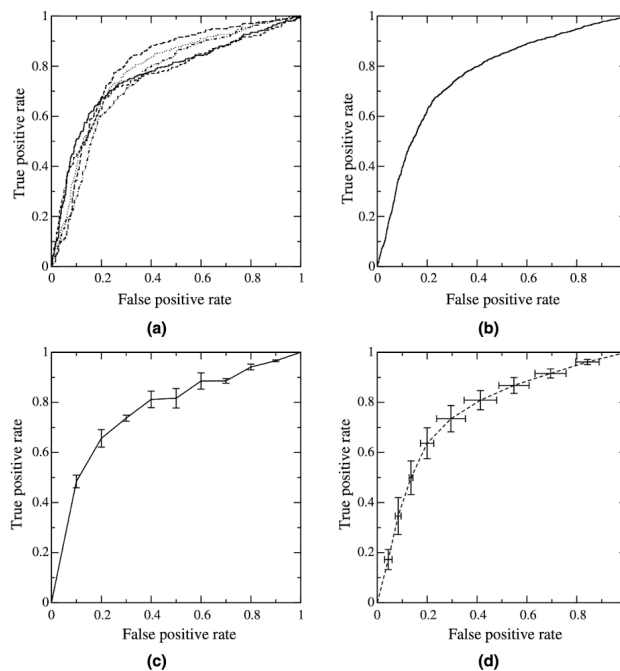
ROC averaging에서 유의점

ROC curve는 classifier의 성능을 평가하는데 사용되지만, 그럼에도 불구하고 classifier의 우수성에 대해서 결론을 낼 때는 주의가 필요하다. 단순히 여러 classifier들의 ROC 그래프

를 보고 어떤 것이 더 우수한지 판단하는 것은 오해의 소지가 있다. 이것은 단지 하나의 test set에 대한 결과로 variance에 관한 측정 없이는 classifier에 대한 비교를 할 수 없다.

original instance를 사용가능하다면 averaging ROC는 쉽게 할 수 있다. CV 또는 bootstrap을 통하여 만들어진 test set T_1, T_2, \dots, T_n 이 있을때 하나의 데이터셋 T_M 으로 병합한 후 할당된 스코어에 따라 정렬한 뒤 결과를 plot으로 나타낸다. 그러나 이렇게 간단히 병합하는 것은 여러개의 테스트 셋을 활용하는 주된 이유인 variance의 측정은 불가하다. 개별의 곡선을 다른 포인트들에서 샘플링하고 샘플들을 평균을 내는 더 정교한 방법이 필요하다.

ROC space는 2차원인 반면 평균은 1차원이다. ROC curve를 1차원에 투영시켜 단순히 평균을 낼 수 있지만 그렇게 투영하는 것이 적절한지 정확한지에 관해서는 의문이다. 이 섹션에서는 ROC curve를 averaging하는 두 방법에 대하여 소개한다 : 1) vertical averaging
2) threshold averaging



- (a) average되어야 하는 5개의 roc curve이며 각각은 1000개의 point로 이루어져있다
- (b) 5개의 test set을 결합하여 그린 roc plot
- (c)&(d) 5개의 개별적인 roc plot을 sampling 하여 그린 roc plot이며 error bar는 95% 신뢰구간을 의미한다.

8.1 Vertical averaging

vertical averaging이란 고정된 FP rate에 대하여 수직의 샘플을 이용해 각각이 대응하는 TP rate의 평균을 내는 방법입니다. 해당 방법은 연구자가 FP rate를 임의로 고정 가능하게

나 single-dimensional variation 측정이 필요할 때 적절한 방법입니다. k-fold cross validation을 위한 classifier들의 ROC curve들을 averaging하는데 활용되기도 하였습니다.

해당 방법에서 각각의 ROC curve는 개별 함수처럼 다뤄집니다. ROC curve 를 함수로 다음과 같이 표현할 수 있습니다.

$$tp\ rate = R_i(fp\ rate)$$

averaged ROC curve는 아래의 식으로 표현할 수 있다.

$$\hat{R}(fp\ rate) = mean[R_i(fp\ rate)]$$

average ROC plot을 생성하기 위하여 \hat{R} fp rate 축을 따라 균등하게 배치된 점들을 따라 \hat{R} 에서 샘플을 뽑을 수 있다. tp rate의 평균의 신뢰구간은 이항분포의 가정을 활용하여 계산될 수 있다. 다음의 Algorithm 3 은 ROC point 집합의 vertical average를 구하는 과정이다.

Algorithm 3. Vertical averaging of ROC curves
Inputs: *samples*, the number of FP samples; *nrocs*, the number of ROC curves to be sampled, *ROCS*[*nrocs*], an array of *nrocs* ROC curves; *npts*[*m*], the number of points in ROC curve *m*. Each ROC point is a structure of two members, the rates *fpr* and *tpr*.
Output: Array *tpravg*[*samples* + 1], containing the vertical averages.

```

1: s ← 1
2: for fprsample = 0 to 1 by 1/samples do
3:   tprsum ← 0
4:   for i = 1 to nrocs do
5:     tprsum ← tprsum + TPR_FOR_FPR(fprsample,
      ROCS[i], npts[i])
6:   end for
7:   tpravg[s] ← tprsum/nrocs
8:   s ← s + 1
9: end for
10: end
1: function TPR_FOR_FPR(fprsample, ROC, npts)
2: i ← 1
3: while i < npts and ROC [i + 1].fpr ≤ fprsample do
4:   i ← i + 1
5: end while
6: if ROC[i].fpr = fprsample then
7:   return ROC[i].tpr
8: else
9:   return INTERPOLATE(ROC[i], ROC [i + 1].fprsample)
10: end if
11: end function
1: function INTERPOLATE(ROCP1, ROCP2, X)
2: slope = (ROCP2.tpr - ROCP1.tpr) / (ROCP2.fpr -
      ROCP1.fpr)
3: return ROCP1.tpr + slope · (X - ROCP1.fpr)
4: end function

```

8.2 Threshold averaging

vertical averaging의 한계점

vertical averaging은 신뢰구간을 계산을 수월하게 하는 single dependent variable인 tp rate를 평균낸다는 장점이 있다. 그러나 independent variable인 fp rate가 연구자의 통제하에 있지않은 경우가 생길 수 있다. 그러한 경우에는 classifier score의 threshold 같이 통제 가능한 변수를 활용하여 ROC point들을 평균내는 것이 선호되기도 한다.

threshold averaging

Threshold averaging은 ROC space에서의 위치에 기반하여 point들을 샘플링하는 대신 vertical averaging과 비슷하게 point들을 도출해낸 threshold에 기반하여 샘플을 추출한다. 샘플을 추출하기 위해서 threshold의 집합이 필요하며 이후에 각 threshold마다 대응하는 ROC curve를 구한 후 평균을 낸다. 아래의 Algorithm 4 는 threshold averaging의 과정을 담고있다.

Algorithm 4. Threshold averaging of ROC curves
Inputs: *samples*, the number of threshold samples; *nrocs*, the number of ROC curves to be sampled; *ROCS*[*nrocs*], an array of *nrocs* ROC curves sorted by score; *npts*[*m*], the number of points in ROC curve *m*. Each ROC point is a structure of three members, *fpr*, *tpr* and score.
Output: *Avg*[*samples* + 1], an array of (*X*, *Y*) points constituting the average ROC curve.
Require: *samples* > 1
1: initialize array *T* to contain all scores of all ROC points
2: sort *T* in descending order
3: *s* ← 1
4: **for** *tidx* = 1 **to** *length*(*T*) **by** *int*(*length*(*T*)/*samples*) **do**
5: *fprsum* ← 0
6: *tprsum* ← 0
7: **for** *i* = 1 **to** *nrocs* **do**
8: *p* ← ROC_POINT_AT_THRESHOLD(*ROCS*[*i*], *npts*[*i*], *T*[*tidx*])
9: *fprsum* ← *fprsum* + *p.fpr*
10: *tprsum* ← *tprsum* + *p.tpr*
11: **end for**
12: *Avg*[*s*] ← (*fprsum*/*nrocs*, *tprsum*/*nrocs*)
13: *s* ← *s* + 1
14: **end for**
15: **end**
1: **function** ROC_POINT_AT_THRESHOLD(*ROC*, *npts*, *thresh*)
2: *i* ← 1
3: **while** *i* ≤ *npts* **and** *ROC*[*i*]. *score* > *thresh* **do**
4: *i* ← *i* + 1
5: **end while**
6: **return** *ROC*[*i*]
7: **end function**

9. Decision problems with more than two classes

two-class analysis의 특징

지금까지는 2개의 class만 존재할 때 분류 성능에 관한 내용을 다루었으며 ROC에 관한 문헌들 대부분은 이러한 가정을 하고 전개된다. ROC 분석은 주로 비정상적인 조건이 있는지 없는지를 다루는 진단의 문제에서 의학적 결정을 내릴 때 사용된다. 두 축은 error(fp)과 benefits(tp)간의 tradeoff관계를 나타낸다. 대부분의 분석은 two-class 문제에 있는 대칭성 때문에 간단하다. 결과 또한 2차원에 그래프로 그려질 수 있으며 시각화하기 쉽다.

9.1 Multi-class ROC graphs

n 개의 class를 분류할 때 혼동행렬의 크기는 $n \times n$ 이며 이 행렬에서 정확한 classification은 n 가지이며 가능한 오류의 수는 $n^2 - n$ 이다. TP와 FP의 tradeoff관계를 다루는 대신 n 개의 benefit과 $n^2 - n$ 개의 error에 대해 다루게된다.

class reference formulation

n 개의 class를 다루는 방법 중 하나는 각각의 class에 대한 총 n 개의 ROC graph를 생성하는 것이다.

$$P_i = c_i$$

$$N_i = \bigcup_{j \neq i} c_j \in C$$

- C : 모든 class의 집합

ROC graph i 는 class c_i 는 positive, 나머지는 negative인 classifier의 성능을 나타내는 그래프다.

그러나 c_i 가 positive일 때 c_i 가 아닌 negative class 내의 분포가 변할 때 문제가 발생한다. c_i 가 아닌 특정 c_k 를 분류하기 쉬운 경우 학습할 때 해당 클래스에 치우칠 수 있다.

9.3 Multi-class AUC

two-class 문제에서 AUC는 하나의 scalar value였지만, multi-class 문제에서는 여러개의 class의 쌍에서 도출된 값들을 결합하는 문제가 있다. 이와 관련해서 더 자세한 사항은 Hand and Till's (2001)을 참고하는 것을 추천한다.

또 다른 multi-class AUC를 계산하는 방법은 각 클래스마다 reference ROC curve를 만들고 curve 아래의 면적을 구한 뒤 reference class의 비중을 가중치로 두어 합하는 방법이다. 이 과정을 아래의 식으로 나타낼 수 있다.

$$AUC_{total} = \sum_{c_i \in C} AUC(c_i) \cdot p(c_i)$$

- $AUC(c_i)$: c_i 의 reference ROC curve 아래의 영역

위의 방식으로 AUC를 구하는 것은 $|C|$ 번의 AUC 계산을 요구하며 전체의 복잡도는 $O(|C|n \log n)$ 이다. 이 방식의 장점은 AUC_{total} 이 class reference ROC curve에서 직접 구할 수 있으며 쉽게 시각화 할 수 있다는 점이다.

마지막 방법은 class distribution과 error cost에 덜 민감한 방법이다. AUC는 랜덤하게 선택된 negative instance보다 positive instance의 확률이 더 높을 가능성과 같다는 사실에

기반한 방법이다. 한 쌍의 class에 대한 분류 성능을 가중치를 적용하지 않고 구한다.

$$AUC_{total} = \frac{2}{|C|(|C| - 1)} \sum_{\{c_i, c_j\} \in C} AUC(c_i, c_j)$$

- $AUC(c_i, c_j)$: c_i, c_j 두 class의 AUC
- 가능한 모든 pairwise에 대해 계산한다


11. Conclusion

ROC 그래프는 분류 모델의 성능을 평가하고 시각화하는데 유용한 도구다. accuracy, error rate 또는 error cost와 같이 단일한 scalar값의 평가 방법보다 풍부하게 분류 성능을 평가할 수 있다. 또한 ROC curve는 class skew 와 error cost를 분류 성능과 분리하여 평가하기에 다른 평가 지표에 비해 유리하다. 그러나 어떤 평가 지표든 현명하게 사용하기 위해선 특성과 한계를 알아야 한다.

▼ 참고

 원투쓰리포

 파이브식스세븐텐

 에잇!나인세븐일레븐